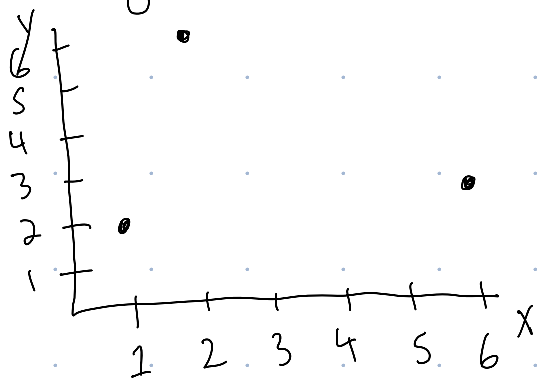
Closest Points Problem

Input: Array of 2-D points: $P = \boxed{(1,2)}\boxed{(2,6)}\boxed{(6,3)} \ldots$



Output: Distance b/t 2 closest points

$\hookrightarrow d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Assume: x, y coordinates are unique for each point

~~| (1,2) | (1,6) |~~

| (1,2) | (2,1) | OK

Applications:
- air traffic control
- robotics
- stereo → 3D

Algorithms + Ethics

Algorithm is essentially a mathematical object.
But once it gets implemented for a particular task, has ethical implications

Ethics of Air Traffic Control Improvement Alg.

0. Where do you want to travel?
1. Who are stakeholders?
2. Who benefits from this implementation?
3. Who is harmed from this implementation?
4. Does this implementation reinforce existing inequities?
5. Would YOU implement?

Closest Points 2D

Before designing a sophisticated algorithm, try to benchmark

- Want better than "Brute Force" $O(n^2)$
- Can't do better than 1-D $O(n \log n)$ Runtime

Brute Force (check every pair)

- min ← ∞
- for i ← 1 to n−1
  - for j ← 1 to n
    - if dist(p_i, p_j) < min, then min ← dist(p_i, p_j)
- return min

Closest Pts 1D

5	9	0	20	17

0	5	9	17	20

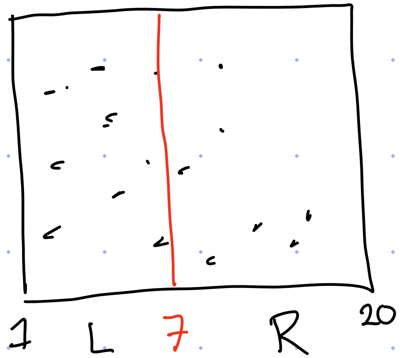
$O(n \log n)$

- Sort pts w/ MergeSort
- min ← ∞
- for i = 1 to n−1 :
  - if dist(p_i, p_{i+1}) < min, then min ← dist(p_i, p_{i+1})
- return min

CloPts(P) (Divide + Conquer 2D Closest Pts)

Base Case: later

Divide:



Sort by x

(1,5)	(2,2)	(4,0)	(10 20)	(12,3)	(20,6)

$\underbrace{(1,5)\ (2,2)\ (4,0)}_{L}$ 7 $\underbrace{(10\ 20)\ (12,3)\ (20,6)}_{R}$

is b/t 7
and 11

Conquer:

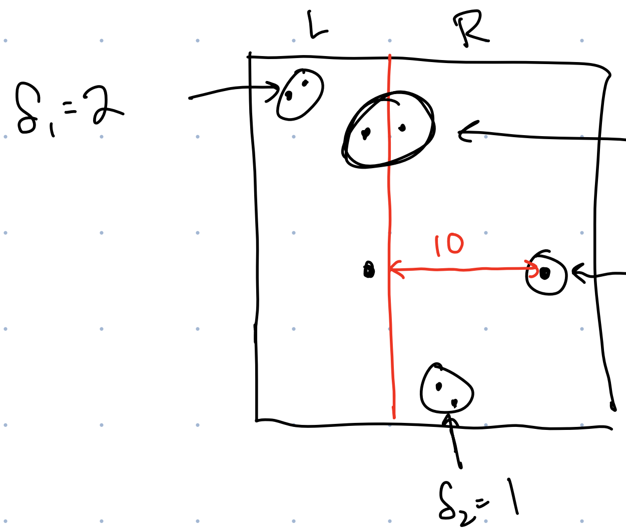
$\delta_1 =$ CloPt(L)

$\delta_2 =$ CloPt(R)

$\delta = \min(\delta_1, \delta_2)$

Combine:

Let's think about this:



Concerned about pairs that cross midline

This pt is too far away from midline to be a concern

What points do we need to worry about in combine step?
Those within

A) $\delta/2$ of midline

C) δ of midline

B) $\sqrt{\delta}$ of midline

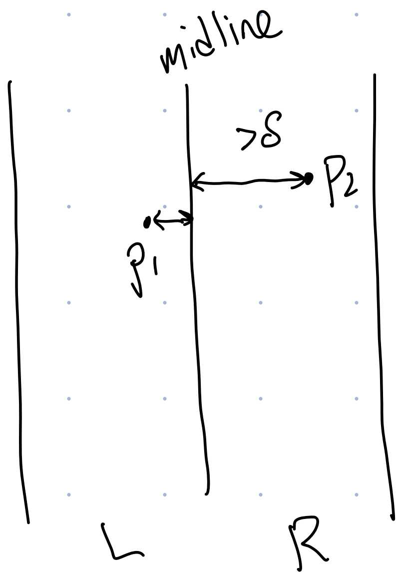
D) 2δ of midline

Recall

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Lemma: If p_2 in R is more than δ from midline, its distance to any point p_1 in L is more than δ.

Proof Sketch: (Basic ideas, but need to add in English explanations in a real proof.)



$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad \geq 0$$

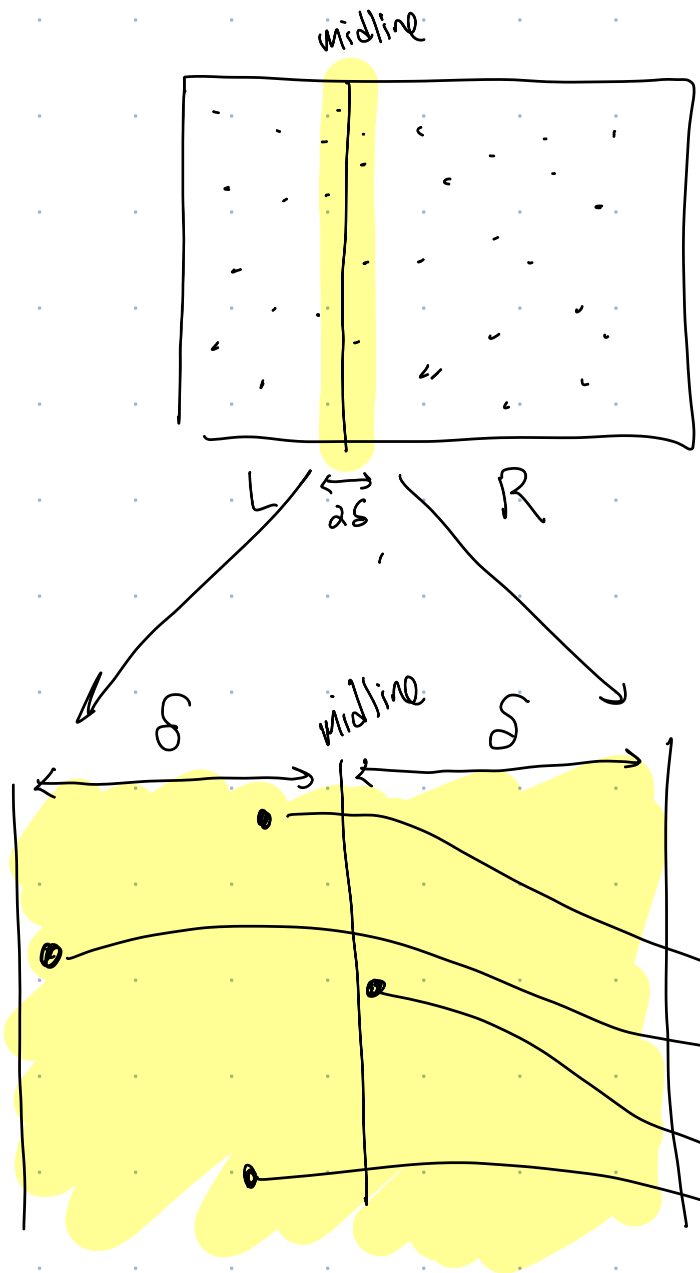
But $(x_1 - x_2)^2 = (\text{distance from } x_1 \text{ to midline} + \text{distance from } x_2 \text{ to midline} > \delta)^2$

$\geq (\delta)^2$

And: $(y_1 - y_2)^2 > 0$

So: $(x_1 - x_2)^2 + (y_1 - y_2)^2 > \delta^2$

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} > \delta$$

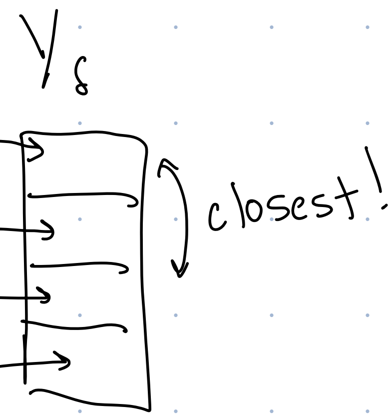


We want to check points in this region,
Looks like a line! Use line approach!

Combine Step

$Y_\delta \leftarrow$ y-sorted list of pts in P within δ of midline

For $p \in Y_\delta$
- Check distance from p to next ___ pts in Y_δ
- Save if smallest distance found



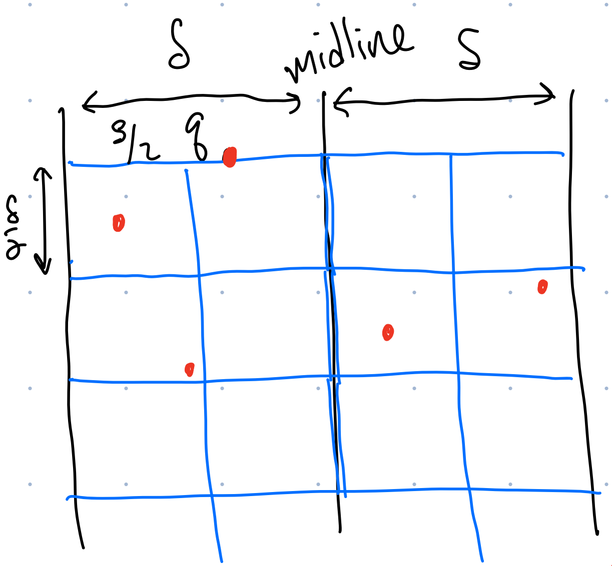
What goes here?

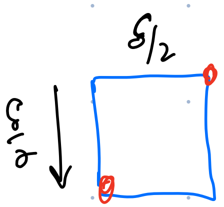
Lemma: Only need to look at next 7 pts in Y_δ

probably not optimal, but easy-ish to prove

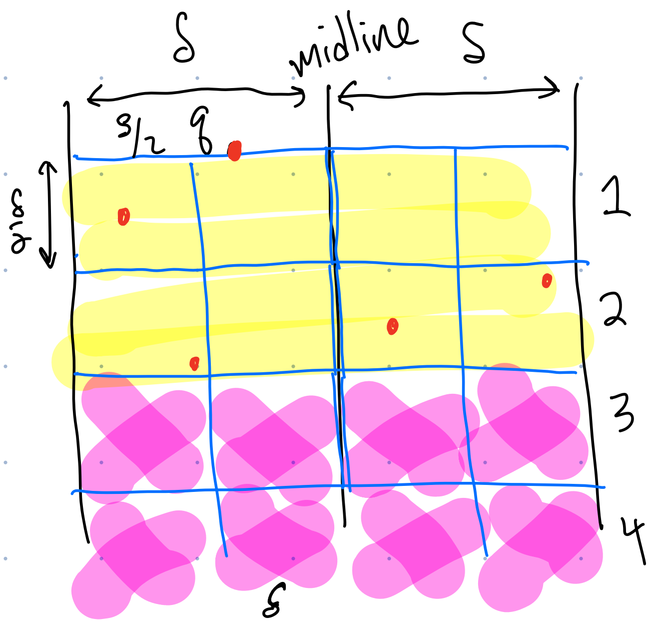
Pf: Imagine dividing region w/in δ of midline into $\frac{\delta}{2} \times \frac{\delta}{2}$ squares starting at current pt (q). Each square can contain at most one point. To see this, for contradiction, suppose there are 2 pts in a square. The pts have largest distance when on opposite corners. In that case, their distance is

$$\sqrt{\left(\frac{\delta}{2}\right)^2+\left(\frac{\delta}{2}\right)^2} = \sqrt{\frac{\delta^2}{4}+\frac{\delta^2}{4}} = \sqrt{\frac{2\delta^2}{4}} = \sqrt{\frac{\delta^2}{2}} = \frac{\delta}{\sqrt{2}}.$$





But each box is entirely in L or R, so any 2 pts both in L or R must have distance at least δ, a contradiction.



Pts in rows 3+ have distance at least δ from q, because difference in y-coord. to q is at leas δ, so we can ignore. So only first two rows of boxes are relevant. Pt q is already in one box, so 7 other boxes can contain at most 7 more points. These points will appear next after q in Y_S, so if check next 7 pts, we will definitely check any pt that might have distance to q less than δ.

CloPts(P) (Divide + Conquer 2D Closest Pts)

Base Case: later

Divide: Sort by x

$$\underbrace{(1,5)\,\big|\,(2,2)\,\big|\,(4,0)}_{L}\ \Big|_{7}\ \underbrace{(10,20)\,\big|\,(12,3)\,\big|\,(20,6)}_{R}$$

Divide into L, R by midline

Conquer:

$\delta_1 = \text{CloPt}(L)$

$\delta_2 = \text{CloPt}(R)$

$\delta = \min(\delta_1, \delta_2)$

Combine:

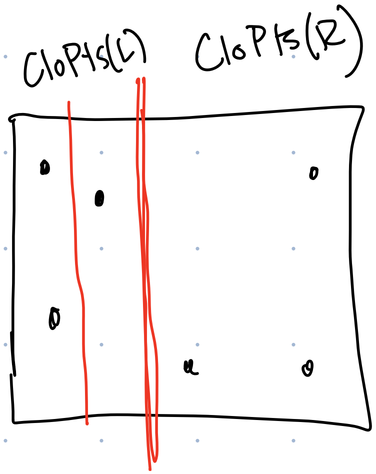
$Y_\delta \leftarrow$ pts w/in δ of midline, sorted by y-coordinate

```
for p_i ∈ Y_δ:
    for j ← i+1 to i+7:
        if d(p_i, p_j) < δ, then δ ← d(p_i, p_j)
return δ
```

Base Case!

What size set of pts should trigger base case?

A) 0 B) ≤ 1 C) ≤ 2 D) ≤ 3



CloPts(P) (Divide + Conquer 2D Closest Pts)

Base Case: If $|P| \leq 3$, then do brute force

Divide: Sort by x

(1,5)	(2,2)	(4,0)	(10 20)	(12,3)	(20,6)

L — 7 — R

Divide into L, R by midline

Conquer:

$\delta_1 = \text{CloPt}(L)$

$\delta_2 = \text{CloPt}(R)$

$\delta = \min(\delta_1, \delta_2)$

Combine:

$Y_\delta \leftarrow$ pts. w/in δ of midline, sorted by y-coordinate

for $p_i \in Y_\delta$:
  for $j \leftarrow i+1$ to $i+7$:
    if $d(p_i, p_j) < \delta$, then $\delta \leftarrow d(p_i, p_j)$

return δ

- Fun weekend plan
- Create recurrence relation for runtime.
- Explain why correct
- Q's about correctness

Problem: Sorting at each recursive step takes too long. Pre Sort!

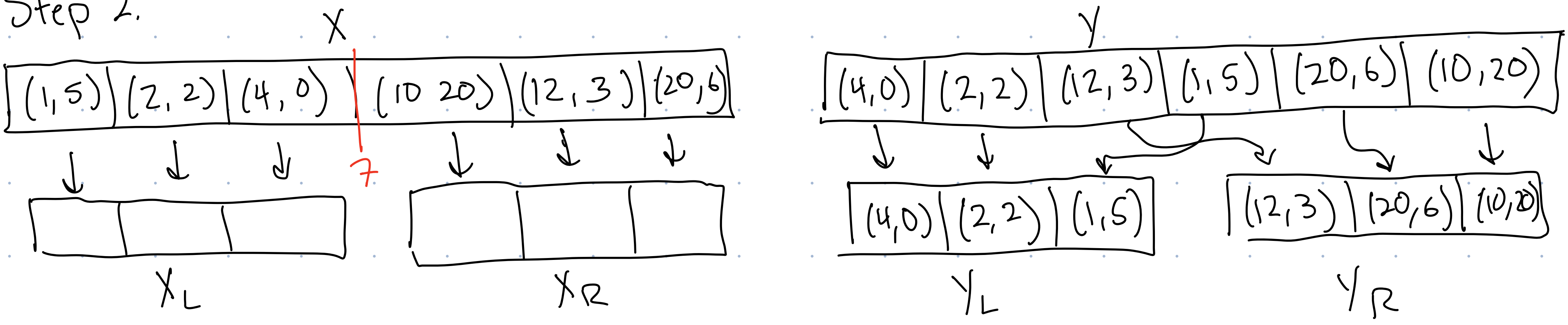
PreSort(P)
- $X \leftarrow P$ sorted by x
- $Y \leftarrow P$ sorted by y
- return X, Y

CloPts(X, Y)
1. If $|X| \le 3$, do Brute Force $O(1)$
2. Divide into X_L, Y_L, X_R, Y_R $O(n)$
3. $\delta = \min\{\text{CloPts}(X_L, Y_L), \text{CloPts}(X_R, Y_R)\}$ $2T(n/2) + O(1)$
4. Create Y_δ $O(n)$
5. For $p_i \in Y_\delta$: $O(n)$
   - For $j \leftarrow i+1$ to $i+7$ $O(1)$
     - if $\text{dist}(p_i, p_j) < \delta$, $\delta \leftarrow \text{dist}(p_i, p_j)$
6. Return δ. $O(1)$

$$T(n) = \begin{cases} O(1) & \text{if } n \le 3 \\ 2T(n/2) + O(n) & \text{if } n > 3 \end{cases} \Rightarrow O(n \log n)$$

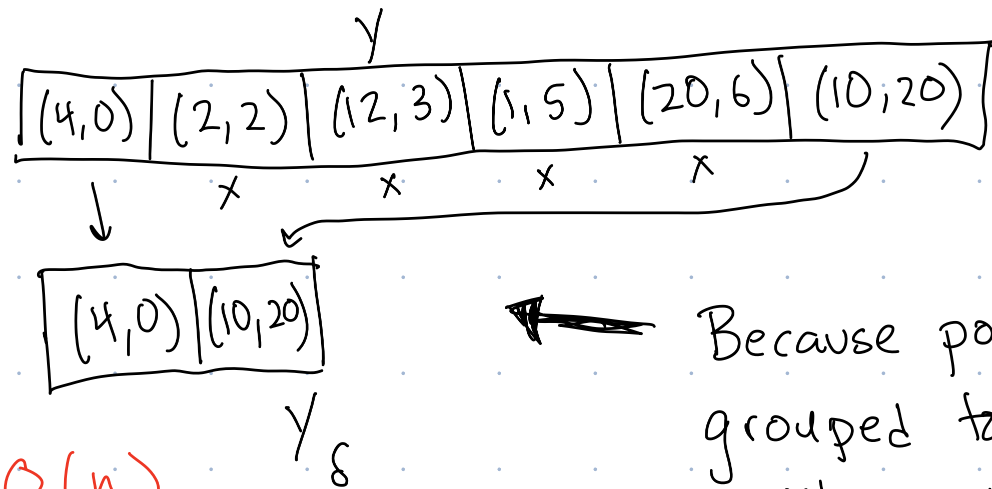
Dividing Arrays Efficiently

Step 2.



O(n)

Step 4 : $\delta = 4$ Want x from 3 to 11



O(n)

Because points in L and R are grouped together in one array, until we look at the pt, we don't know if in L or R

- Where did we use our assumption that x,y points are unique?

  Difficult to divide into L and R using $O(n)$ time.

- You can do Prg. Assign. 1!